Abstract

Currently there is much hype surrounding web services. Essentially web services are a simple platform neutral means for programs to communicate across the Internet. They promise to bring distributed computing to the Internet and to pick up where DCOM, RMI and CORBA technologies have failed. Web services can represent messaging or RPC style communication; however they are considerably more restricted than traditional communications technologies. This paper describes a model for web services that incorporates object oriented features, including the encapsulation of state, object references, and object migration. In this way web services achieve similar flexibility to distributed object technologies such as RMI and achieve other important goals such as simple deployment. All this is done without needing any new technologies such as languages or protocols, or needing to extend web services.

1. Introduction

Web services represent distributed computing for the Internet, and promise to enable the next generation of the Internet [6, 7]. Web services are designed as a simple platform neutral mechanism for program to program communication across the Internet. Web services typically involve XML based messaging, often encoding RPC. They have some similarities with CORBA, DCOM, Java RMI etc. but whereas these systems are primarily designed for homogeneous client server computing on local area networks, web services are designed for heterogeneous communication across the Internet. (Although CORBA, and other middleware, was designed for heterogeneous communications, in practice this was rarely achieved.) There is now widespread support for web services in the industry.

Unfortunately the simple design of web services which has contributed to their popularity is also a weakness; web services do not give you much, just basic message passing between endpoints. Currently there are efforts underway to incorporate security, transactions, routing and other important features into web services e.g. Microsoft’s GXA effort [6]. There is also some work towards architectures for web services for large e-commerce sites using n-tier architectures and the like [4, 5]. However there has been little work on refining the model of web services. A notable exception is the work by Fielding [1], who has reverse engineered the architecture of the web; this is discussed in Section 5 Related Work.

In this paper a model is proposed for constructing distributed web service based systems. The model overcomes some limitations in the way that web service systems are usually constructed. The model supports the simple and natural construction of web based distributed systems. It has a natural object oriented style, supports simple deployment, is dynamic, and yet is purely web service based. It loosely follows the REST [1] principle.

The remainder of this paper is organised as follows. Section 2 overviews XML and web services, and some deficiencies in how web services are typically used. Section 3 outlines the proposed model of web objects. Section 4 describes a simple implementation, Section 5 covers related work, and Section 6 discusses initial findings and further research directions.

2. XML and web services

XML is a data description language. It supports the description of semi-structured hierarchical data. XML comprises elements and attributes. Elements can contain text; they may also be composite i.e. contain further elements and attributes. Attributes are atomic and represent atomic data. XML may be typed though XML schemas. These enable guarantees to be made about data attributes and elements being present, in the right number, and having the right form. Thus we may guarantee that, for example, any XML data claiming to represent a person has a name, an address and optionally an email address. XML is often com-
pared with HTML; essentially XML describes content and HTML describes presentation (although not originally intended so). XML must be syntactically well formed and may require conformance to a schema; it is precise; there is no concept of trying to recover from badly formed XML, unlike web browsers’ lax interpretation of HTML. The most important attributes of XML are that it has been standardised and that it has widespread adoption.

Web services represent the next generation of the Internet. Much simpler than CORBA, Java RMI and DCOM they represent a simple form of messaging or RPC designed for communication between heterogeneous systems across the Internet. Typically they operate on top of the HTTP protocol, and are encoded using XML. The most common web service protocol is SOAP - Simple Object Access Protocol [14]. SOAP supports both RPC and messaging style communication using XML. Messages may be either typed or untyped (arbitrary XML). The Web Service Description Language is analogous to IDL; it is an XML description of web services: web service metadata. It used by tools to generate communication proxies and stubs, and by browsers in IDEs.

The conventional architecture for web services is to incorporate them as communications end points which interface into the business logic layer of an n-tier system. This provides a programmatic interface in addition to the standard web based interface of such systems, see Figure 1.

![Figure 1. Conventional web services architecture](image)

However there are a number of disadvantages to this approach: such web services are static and rather heavy weight. Typically they implement a fixed number of RPC communications endpoints. (Note, some people advocate using untyped messaging style web services in order to overcome these deficiencies, but this merely elevate problems to another level since web service discovery and meta data are no longer useful and must be implemented at another level.) Thus such web services do not naturally map to object oriented concepts, they are static, and state is not encapsulated e.g. session information is usually implemented separately from the web service. Deployment is also usually a separate heavy weight operation, and user interface generation is separate from web services. The web object model aims to overcome some of these limitations.

3. Web objects

The basic idea of web objects is to utilise a model similar to Java RMI and other distributed object technologies for web services. Essentially web services are rather like static methods. Thus using standard object oriented patterns we can support an object oriented model for web services as follows.

3.1. URIs = object references

URIs are analogous to object references, they reference resources on the web. This is rather profound and is espoused by Fielding [1]. This is already a departure from traditional web services, and even dynamic web pages, where URIs represent communication endpoints. Having URIs denote objects, c.f. web pages, is much more in keeping with the original design of the web. Note we are not suggesting that each URI represents a separate communications endpoint (e.g. servlet or traditional deployed web service), merely that URIs address objects. Unlike Fielding we aim to use web services as the communications protocol, and not existing HTTP operations, such as PUT, GET etc.

URIs representing web object addresses always represent the same object, and object state is encapsulated. In particular there is no notion of implicit session state, different sessions are represented by different objects with consequently different URIs.

3.2. Factory objects, events, and other design patterns

Given that web services are methods and URIs denote objects, how are new objects created? by using factory objects. Thus factory objects create new objects at new URIs, thereby dynamically creating web services. In fact we may use other design patterns such as observer patterns, events and listeners etc. to support a rich object oriented model, rather like Java Beans [8].

3.3. Efficiency

As previously mentioned web object addresses (URIs!) are not necessarily directly implemented as communications end points. Furthermore inter-object communication within the same machine can be optimised, there is no need to use web services for local calls. Such an optimisation can be done dynamically based on the URI used for the web
service. Clearly web objects are coarser grained than traditional objects. In fact each addressable object will typically be implemented by a cluster of programming language objects one of which will act as the exposed object. Such clusters are treated as a single web object c.f. ProActive [12]. There is also an issue of object lifetime. We envisage that some objects will be short lived and others may remain for ever - like static web pages. A form of soft state management needs to be employed to manage objects, and web services can be used to control this e.g. to renew an objects lease, or to inform clients of a lease expiry.

3.4. User interface

It is desirable for web objects to implement lightweight user interfaces such as forms, and then web objects may be used to implement workflows, and other applications. A web object may produce a user interface by supporting a well known web service which produces e.g. XHTML. This can call back to the web object to support interaction, again using web services. Thus the web object becomes an active document which supports editing and automation. This has some similarities with Microsoft’s newly announced XDocs [3]. Another possibility, once it is available, is to use XForms [11], for the user interface.

3.5. Metadata and discovery

Meta data is the key to modern IT. Mechanisms are needed to describe and discover web objects. WSDL the web service description language can be used to describe the services web objects offer; a group of web objects all belonging to the same class will all produce the same WSDL except that the service endpoints will differ. Like many web service implementations, descriptions need to be dynamically generated. The counterpart for discovering web services is UDDI [10]. However UDDI registries are not designed to be updated frequently. For local discovery IBM and Microsoft’s WS-Inspection standard could be used, but this is still under development. A simple solution is for factories and factory builders to be registered with UDDI, since we do not expect these to change frequently. These in turn can support a query interface to query available objects.

3.6. Migration and factory builders

Since state is clearly partitioned between web objects, state may easily be serialised to support object copying and migration between machines. We have said little about the implementation of web objects. This is discussed in the next section, briefly our current implementation uses Java. Thus objects may also be migrated to, deployed, on machines without the object implementation by copying the appropriate Java byte code along with the serialised objects, rather like RMI code servers. Again this is handled by well known web services which represent factory builders. These can create object factories; providing they have the necessary bytecode. Thus deployment becomes a simple operation of invoking a factory builder web service supplying it with bytecode as an argument.

3.7. Summary of the model

The model is summarised below and in Figures 2 and 3.

- Web objects encapsulate object state, they can behave like documents.
- Web objects are referenced by URIs.
- Web object may offer both user interface and programmatic interfaces through web services.
- Web objects may be created, copied, moved, and migrated using web services.
- Web objects may generate and listen for events, which are web services.
- Web object factories may be created by factory builders leading to simple web object deployment.

4. Implementation

A proof of concept implementation of web objects has been developed using Java and the Apache AXIS implementation of SOAP web services [9]. The implementation supports web objects through some modifications to the implementation of AXIS. This supports the dynamic creation of web objects, and their referencing. Essentially each class of web objects is represented by a deployed web service.
which maps referenced web objects to their Java counterparts. Thus effectively each class acts as a communications endpoint. An explicit collection of web objects are contained within each class. Methods exist to map between web object URLs and their Java implementations.

HTML is produced by a message based web service. Unfortunately this must be located at a different URL due to current Axis restrictions. By using a document style web service XHTML is produced for consumption by a web browser. An XSL processing instruction in the XML causes the browser to disregard the SOAP web service envelope and to just see the embedded XHTML message. Using URL query parameters to denote methods, form data can be posted back to particular web services. Whilst not particularly elegant, these techniques suffice for generating HTML interfaces.

Using the implementation several toy applications have been constructed to trial features and ideas.

5. Related work

It is useful to compare web objects with Java RMI, probably the best known distributed object system. Java RMI can operate over the HTTP protocol, and hence is not limited to just intranets. However there are different mechanisms to achieve this so interoperation can be complex. An obvious difference between web objects (services) and RMI is that RMI requires a server running RMI and a JVM, it is not designed for cross platform communication. Web services also differ in having high level discovery mechanisms such as UDDI.

Representational State Transfer (REST) is an idea devised by Roy Fielding [1]. It is a model for the web and web services where all resources are referenced by URIs and standard HTTP operations such as PUT and GET are used to implement web services. Web objects are hybrid approach using URIs to denote resources (objects) but using SOAP like web services to encode operations.

In the Grid computing community two groups have been looking at web services: the Extreme Computing group at the University of Indiana, XCAT [2] and the Globus Project’s Open Grid Services Architecture (OGSA) [13]. The XCAT project has a component programming model for Grid web services. They use a scripting language to compose web services, particularly web services which comprise wrapped legacy applications. Web services are also used to implement application factories for creating and managing application instances. The OGSA initiative is a Grid infrastructure based on web services - an evolution of Globus. It is a large initiative; it defines Grid services and interfaces for discovery, dynamic service creation, lifetime management, notification, and management. Both systems have similarities with web objects; however they are primarily designed for wrapping legacy applications for composition using web services. Web objects are designed to operate at a finer granularity, are more object oriented, support user interfaces, and address deployment issues.

Recently Microsoft has announced an addition to the Office suite of programs called XDocs [3]. This is a rich forms based system which communicates with a backend database using web services. The use of web services and documents is similar to the idea of using web objects to represent forms.

6. Discussion

The proposed model of web objects achieves a natural model of distributed computing for the web based on objects. It represents a hybridisation of web services and distributed objects. Compared to traditional web services it is more dynamic, more object oriented, and supports migration. By utilising web services the model is also platform neutral except for the migration of program code (Java classfiles).

The model of web objects seems very different from the traditional n-tier model with data stored in a back-end database, and accessed through layers of business logic etc. However objects themselves can still be structured in a layered fashion. It is even possible to store web objects in a database if desirable.

The current implementation has a number of shortcomings. It does not recycle objects. Clearly some form of soft state management is desirable in a serious implementation. Also there is also no security or grouping of objects into applications or components. Effectively in the current implementation objects are grouped by class e.g. URLs of...
the form “/class/object”. It is desirable to group objects in other ways e.g. by application or owner. There is also an issue of threading, objects are single threaded which can limit concurrency. Objects may invoke web services from within web services. Currently the restriction is that such web services must either be one way or at least of short duration and certainly non-blocking.

The use of HTML for the interfaces is rather clumsy a better solution would be to use XForms for form based interaction - note at present this is still being standardised by the W3C. Also strictly speaking the use of an XML processing instruction, for generating the user interface, is not SOAP compliant.

Using Java as the programming language has the advantage that implementations are available for a variety of machines and code (class files) can be copied between machines to support object migration (the current implementation does not support this though). A downside is that web object interactions are indirectly achieved through Java. Also the programmer now has to deal with different kinds of objects, those exposed as web objects and those just used for their implementation. This is not that different from RMI etc. but it is a little clumsy. Other languages could be used for the implementation such as Microsoft’s C# or ECMA Script. An intriguing possibility is to use XSLT or XQuery, thereby eliminating any impedance mismatch between the programming language and web services (XML).

In general to seriously use web objects a major revision to a web service implementation such as Axis is required. We are currently investigating how best to do this, and the features needed for more realistic applications.

7. Acknowledgements

This work was carried out whilst the author was on sabbatical at INRIA Sophia Antipolis.

References